THE CONNECTION MACHINE™ COMPUTER

A More Detailed Look

THE CONNECTION MACHINE™ COMPUTER
A More Detailed Look

The Connection Machine system operates by receiving a stream of instructions from the front end, expanding each of these into a series of machine instructions, then broadcasting these machine instructions, one at a time, to all processors in parallel. The instructions coming in from the front end are referred to as "macro-instructions." The instructions broadcast to the individual processors are called "nano-instructions."

The Connection Machine system has 65,536 physical processors, but may be configured for a much larger number of logical processors by means of the *global-reset* and *configure* commands. *Configure* takes two arguments, allowing a two dimensional array of virtual processors per physical processor. *Configure 4,4*, for example, sets up the machine in the million-processor mode. (Or, more precisely, the 1,048,576 processor mode.) The same number of virtual processors could be established by the command *configure 16,1*. Since virtual processors are so commonly used on the system, they will be referred to simply as "processors." Where it is necessary to refer to one of the 65,536 hardware processors, the term "physical processor" will be used.

Each physical processor has 4096 bits of memory, making 32 Mbytes for the machine as a whole. In the million-processor mode, each processor has 256 bits of memory. Memory is divided into a data area and a stack area, with the layout being the same in each processor. A single, system-wide register, the stack limit, defines the boundary between stack space and data space. The stack pointer is also a system-wide register. The stacks in all processors act in unison.

Memory is bit-addressable; fields can be of arbitrary length. There are three standard numeric formats: unsigned integer, signed integer, and floating point. Each is of arbitrary length. In particular, floating point numbers can be of any length. (If they are 32 bits or 64 bits long, they conform to the IEEE standard.)

Addresses in the system as a whole operate at three levels. The first is the physical processor level. Corresponding to

each physical processor, there are one or more virtual processors. Corresponding to each virtual processor, there are a number of bits of memory.

Data may be exchanged between the Connection Machine memory and the front end in any of three ways. *Read-slice* reads a single bit of information from the memory of each of a series of consecutive processors, assembles them into a signed integer, and passes the integer to the front end. *Write-slice* goes the other way. Slice operations are typically done 16 or 32 processors at a time. *Read-processor* and *write-processor* move a single field between the front end and a single processor. *Read-array* and *write-array* move fields between the front end and a set of contiguous processors.

All instructions flow into the Connection Machine hardware from the front end. These macro-instructions come into a microcontroller, which expands them into a series of nano-instructions. Some expand into just a few nano-instructions. Others expand into as many as a hundred or more. It is also possible to feed nano-level instructions to the microcontroller and control the hardware directly. It is not, however, efficient to do so, because the front-end cannot supply these instructions rapidly enough to keep the system busy. (Direct control of the hardware from the front end is provided primarily so that the front end can support debugging and diagnostic aids.)

Nano-instructions are broadcast to all virtual processors. Processors, however, have the option of "sitting out" a series of instructions. A one-bit flag within each processor, the Context Flag, determines whether that individual processor will respond to the instruction or not. Most of the instructions discussed below are "conditional" in the sense that they only take effect in the processors whose Context Flag is 1.

The Connection Machine system is implemented with four physical microcontrollers, one for each 16,384 processors. If the system has a single front end, that front end is connected

to all four microcontrollers and therefore drives all 65,536 processors in parallel. A system may be configured with up to four front ends. A switch called the Nexus makes the connections between front ends and microcontrollers. It is possible, for example, for four users to be operating simultaneously. Each works at a separate front end, and each has a separate instruction stream executing in a quarter of the system's processors. For the remainder of this document, it is assumed that the system is operating with a single front end.

Processor Instructions

The system's computational macro-instructions will be familiar to the user of any modern macro assembler. They operate on signed integers, unsigned integers, and floating point values. They include unary operators such as *not*, *negate*, and *absolute-value*. They include binary operators such as *and*, *or*, *add*, *subtract*, *multiply*, *divide*, *compare*, *shift*, and *move*. What is unique about these instructions on the Connection Machine computer is that they operate on every selected processor in parallel. The results, of course, are different in each processor, because each is doing the operation on its own data.

In the case of the *random* instruction, this difference is explicit. *Random* places an independently chosen random number in each selected processor. Two processors may or may not be assigned the same random value. The *enumerate* operation, on the other hand, guarantees to assign a distinct integer to each selected processor.

Global instructions produce a single result from data items stored in the memories of all selected processors. *Global-logical-or* for example, takes the inclusive or of a field in each processor's memory. *Global-count* examines a single-bit field in all processors and returns the number of "1" bits. *Global-add* sums multi-bit fields. *Global-max (-min)* returns the largest (smallest) value found in a specified field across all selected processors. *Global-add* and *global-count* operate on unsigned integers, signed integers, and floating point values, as does *global-max (-min.)*. The *enumerate* instruction

goes in the other direction, placing a different consecutive integer into each of a selected range of processors.

A Connection Machine processor is a single-bit computer which operates on two data values, A and B. Each processor also has an array of flags which can participate in operations. Instructions specify the address of A (within the processor's own memory), the address of B, and the number of the flag to be used. Results are always stored back into A. A flag value may be stored as well. Since an instruction at this level involves three input bits, there are 256 possible Boolean operations. All 256 are implemented.

The simplest form of communication between Connection Machine processors is between nearest neighbors. Each processor is wired to its neighbors to the North, East, West, and South by a communications grid called the NEWS network. Four instructions, *get-from-north*, *get-from-east*, *get-from-west*, and *get-from-south* control the transfer of data. NEWS information is passed one bit at a time.

Beside the NEWS grid is a much more powerful communications system, the Connection Machine Router. It allows full messages to be sent from any processor to any other; the sending processor simply needs to know the address of the destination processor. Messages may be of any length. Typical messages contain 32 bits of information; adding the address information and headers results in a transmitted package of 52 bits.

Each of the 65,536 physical processors is connected to 16 other physical processors in a special organization called a "Boolean n-cube." Because 65,536 is expressed in 16 bits in binary, the Connection Machine's cube is a Boolean 16-cube. Each of the processors is connected to every other processor whose binary address is different in just one of the 16 bits. The following example shows the interconnections of processors 6_{10} and 2070_{10}. The binary addresses are shown in parentheses.

These two sets of addresses have a common connection. Processors 6 and 2070 both connect to 22. Thus it is possi-

2	(0000 0000 0000 0010)
4	(0000 0000 0000 0100)
6	(0000 0000 0000 0110)
7	(0000 0000 0000 0111)
14	(0000 0000 0000 1110)
22	(0000 0000 0001 0110)
38	(0000 0000 0010 0110)
70	(0000 0000 0100 0110)
134	(0000 0000 1000 0110)
262	(0000 0001 0000 0110)
518	(0000 0010 0000 0110)
1030	(0000 0100 0000 0110)
2054	(0000 1000 0000 0110)
4102	(0001 0000 0000 0110)
8198	(0010 0000 0000 0110)
16390	(0100 0000 0000 0110)
32774	(1000 0000 0000 0110)

Figure 1:
The sixteen processors that are connected to processor 6_{10}.

22	(0000 0000 0001 0110)
2054	(0000 1000 0000 0110)
2066	(0000 1000 0001 0010)
2068	(0000 1000 0001 0100)
2070	(0000 1000 0001 0110)
2071	(0000 1000 0001 0111)
2078	(0000 1000 0001 1110)
2102	(0000 1000 0011 0110)
2134	(0000 1000 0101 0110)
2198	(0000 1000 1001 0110)
2326	(0000 1001 0001 0110)
2582	(0000 1010 0001 0110)
3094	(0000 1100 0001 0110)
6166	(0001 1000 0001 0110)
10262	(0010 1000 0001 0110)
18454	(0100 1000 0001 0110)
34838	(1000 1000 0001 0110)

Figure 2:
The sixteen processors that are connected to processor 2070_{10}.

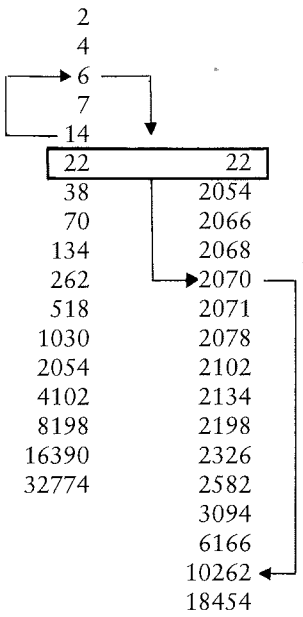

Figure 3:
A four step routing processor.

From Processor	With Relative Address	To Processor	
14	(0010 1000 0001 1000)	6	(move −8)
6	(0010 1000 0001 0000)	22	(move +16)
22	(0010 1000 0000 0000)	2070	(move +2048)
2070	(0010 0000 0000 0000)	10262	(move +8192)
10262	(0000 0000 0000 0000)	---	(done)

Figure 4:
Using relative addresses to compute the routing path.

ble to pass a message, for example, from processor 14 to processor 10262 in just four steps. The router at processor 14 passes it to the router at processor 6, which passes it to 22. From there it goes to 2070 and then to 10262.

A four step routing process

The illustration above was worked out using absolute processor addresses. In fact, the router uses relative addresses. Instead of saying, "Send this message to processor 10262" processor 14 actually says, "Send this message to the processor whose relative address is 10264." This relative address is obtained by XOR'ing the source and destination addresses. The XOR of 0000 0000 0000 1110 and 0010 1000 0001 0110 is 0010 1000 0001 1000. This relative address does more than just indicate the destination. It defines the set of possible routes. Each "1" bit marks a required routing step. The right-most "1" bit is in the bit position whose value is 8_{10}. This bit tells the router to route the message through the chip which is "down 8," in other words, processor 6. At the same time the router sends the message, it clears this bit, so that processor 6 receives a message that says "Send this message to the processor whose relative address is 0010 1000 0001 0000." The destination processor is still 10262. The pattern continues until the message arrives. At this point the relative address is zero, indicating that no more moves are needed.

The routing steps indicated by a relative address may be done in any order. In the simplest case, the 1-bits indicating the required routine steps for a message are processed in right-to-left order. If there is too much traffic along a particular path, however, some messages will take their routing steps in a different order, thus guiding them along alternate paths.

Connection Machine physical processors are grouped sixteen to a chip. There is a single router on each chip that services all sixteen processors. Hence four of the sixteen routing connections are internal to an individual chip. As a result, it takes a maximum of twelve steps to move from any

chip to any other chip. During message routing, the system goes through all twelve steps. If the router on a given chip has a message whose relative address has a "1" in the low order bit position, it sends that message on the first of the twelve steps to the chip whose address differs in that same bit (i.e., the next chip). If the message it contains has a "0" in the low order relative address bit, it doesn't send anything on that step. The process continues through all twelve steps, with all router chips responding in the same way.

The basic message passing instruction is *store*. *Store* specifies the length of the message as well as the relative address of the destination processor. *Store* is a parallel instruction; all selected processors initiate message transfers at once. Special routing hardware handles the volume of messages efficiently. An individual router on a chip may receive as many as 12 messages from other chips during a message cycle, one from each other chip that it is connected to. It can in turn send as many as 12 messages, one on each of the wires. If two messages need to go down the same wire, one is buffered until the next routing cycle. Each of the 4096 routers can buffer as many as 7 messages. If an individual router becomes extremely busy, it can defer acceptance of any new messages from its own processors. This deferral keeps the router free to handle messages from other chips. If the chip's buffer space still fills, it refers messages to neighboring chips.

Parallel message sending also introduces the possibility that the same location in the same processor will receive two or more messages in the same cycle. The simple *store* instruction gives unpredictable results in this case. Several variations of the *store* instruction, such as *store-with-add*, deal with this possibility. If two or more *store-with-add* messages arrive at the same destination, they are summed.

Router throughput has been measured at 3 gigabits per second under heavy load. The test routine repeatedly sent 32 bit messages from all 65,536 processors in parallel. Destination processors were chosen at random, but each message always had a unique destination.

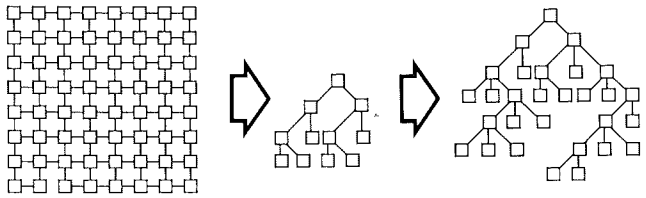

Figure 5:
A machine vision application requires a grid structure for initial pixel analysis, then a series of tree structures as the system works to identify the objects detected. The changes are made dynamically, simply by changing addresses in individual processors.

Dynamic Reconfiguration

A processor address is all it takes to establish a linkage on the system. This flexibility allows applications to reconfigure on the fly. A number of instructions support this capability. The *my-address* instruction allows processors to determine their own addresses, so they can send them to other processors and thus establish new connections. The *processor-cons* instruction allows each selected processor to find another "free" processor.

Processor-cons specifies the address of a one-bit field, the "free flag." A processor is considered free if it has a "1" in that field. The system looks in parallel for processors with 1's and passes the address of one to each selected processor, clearing the bit at the same time.

A machine vision application requires a grid structure for initial pixel analysis, then a series of tree structures as the system works to identify the objects detected. The changes are made dynamically, simply by changing addresses in individual processors.

Programming the Connection Machine

The Connection Machine programming languages deliver the power of parallel processing in a natural and familiar way. REL-2, CM Lisp™, and C* ™ will be instantly familiar to assembly language programmers, Lisp programmers, and C programmers respectively. In each case a very compact set of parallel operators is added to access the massively parallel architecture. The REL-2 and CM Lisp languages are well-defined and are discussed below. Details of the C* language extensions will be released by Thinking Machines Corporation as soon as they are available.

REL-2 is an assembly level language for the system. Its primary role is as a language in which to develop CM Lisp and C*, but it is available for independent use. The REL-2 user interface on the front end consists of a set of Lisp macros and functions to be called from Lisp code. These macros

and functions then direct the actions of the system by sending directives to the Connection Machine microcontroller. The previous section of this document gives the flavor of REL-2. All the italicized commands used as examples in that section are REL-2 commands.

CM Lisp will be immediately familiar to the Lisp programmer. Program flow and control are essentially the same as in a serial Lisp. Parallel operations utilize a data structure called a xector (pronounced zek′ tor.) Xectors allow values to be stored one per processor, allowing simultaneous operations on all the elements. A xector is defined by three components: a domain, a range, and a mapping between them. Each element in the domain is called an index and must be unique; each element in the range is called a value. Stored one per processor, the index functions as the name of the processor. In the xector, {sky→blue grass→green apple→red} sky, grass, and apple are the indices. Blue, green, and red are the values. If the domain of a xector is the sequence of integers starting with zero, the xector may be written in a more compact form. {0→A 1→B 2→C 3→ D} is equivalent to [A B C D].

Because the Connection Machine system operates in parallel on xectors, many of the generic Lisp sequence operations speed up dramatically when used with them. Operations such as SEARCH and DELETE, which can be performed on each element independently, execute in a fixed time no matter how many elements are in the xector. Operations which involve reducing, counting, or numbering the elements take place in logarithmic time, because they are implemented by algorithms on balanced trees.

The Greek letter α is used by CM Lisp to indicate that an operation is to be done many times in parallel. One informal way to think of α is that it means "give me a zillion" of whatever is inside the expression, where "a zillion" is however many are needed. The α notation will produce a zillion additions, a zillion threes, or whatever. For example, (α + {a→1 b→2}{a→3 b→3}) evaluates to {a→4 b→5}. In this case "a zillion times" means twice, because there are just

two elements in the xectors. The α may be factored out of the expression for clarity. Thus α(+ 1 2) is equivalent to (α+ α1 α2).

The Greek letter β is used to define a second kind of parallel operation. β converts a two-argument function into a function that reduces the elements of a xector into a single value. This reduction is performed in logarithmic time. β reduction uses only the values of the element and ignores the indices. For example, (β + {A→1 B→2 C→3}) evaluates to 6. (βMAX [1 3 5 7]) evaluates to 7. An extended form of β can perform many such reductions in parallel.

These are the primary constructs that have been added to the CM Lisp language. In terms of the hardware, α broadcasts single values and performs parallel independent computations; β gathers information together and performs general interprocessor communication.

Summarizing the Connection Machine System

Every element of the Connection Machine system contributes to its applications orientation. Everything focuses on making large-scale problem solving easier. Dynamic reconfiguration sets the computer up the same way the problem is set up. Virtual processors allow a 1-on-1 matching of problem elements and processors. 65,000 physical processors provide the raw computing power (in excess of 1000 MIPS) to solve giant problems quickly. The speed of the router (over 3 billion bits per second) accommodates problems with massive amounts of intercommunication. And yet all this power is delivered with just a handful of new constructs in CM Lisp and C*.

Thinking Machines Corporation believes that advances in large scale computing require both power and simplicity. The power gives access to the needed detail. The simplicity allows the user to forget about the components of the computer and focus on the components of the problem. The magic of the Connection Machine system is that it provides both.

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts
02142-1214

(617) 876-1111